PnG: Micro-structured Prune-and-Grow Networks for Flexible Image Restoration

Wei Jiang, Wei Wang, Shan Liu, Songnan Li
Media Lab, Tencent America LLC.
{vwjiang,rickweiwang,shanl,sunnysnli}@tencent.com

Abstract

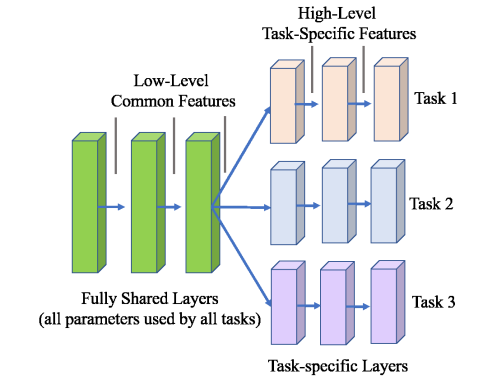
This paper addresses one major issue of DNN-based image restoration: the difficulty of using one model to fit multiple reconstruction requirements, such as supporting different compression ratios in neural image compression (NIC) or different zooming scales in single image super-resolution (SISR). Instead of training an independent model instance for each requirement as an individual task, we develop a practical solution that uses one model instance to support multiple requirements. We propose a general multi-task learning framework based on a novel prune-and-grow (PnG) process, where each task corresponds to each of the requirements. Different from traditional multi-task networks that use fully shared or task-specific layers, we enable in-layer partial parameter sharing to obtain both common and task-specific features at various abstraction levels. This encourages adequate sharing to improve the overall multi-task performance. The parameters are shared at a micro-structured level to both maintan the task performance and reduce inference computation. The sharing structure is automatically learned, where a model instance trained for previous tasks is progressively pruned and regrown to perform more tasks. The framework is task-generic and model-structure-agnostic. Using NIC and SISR as two example applications, extensive experiments show that the multi-task PnG network can largely reduce the overall model size and inference computation, with almost no degradation of the reconstruction performance.

1. Introduction

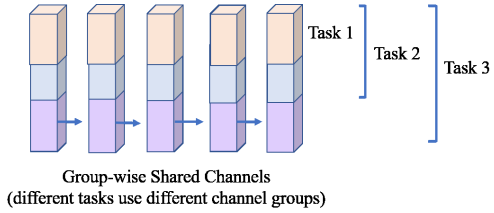
Recently, great sucess has been achieved by using deep neural networks (DNNs) on a variety of image restoration problems, such as neural image compression (NIC) [3, 4, 26, 32], single image super-resolution (SISR) [24, 35, 42, 46], and image denoising [6, 12]. Yet, one main challenge, particularly critical for image restoration, is the flexibility of using one model to fit multiple requirements. For example, NIC methods need to provide different levels of compression ratio based on the system bandwidth and reconstruction quality. SISR methods need to support different zooming scales (*e.g.*, 2×, 3×, 4×) according to user preference or hardware limitation. Image denoising methods need to support denoising at different ISO/gain settings. While one can certainly train one separate DNN model for each of these requirements, it is often too expensive to be practical, in terms of memory, computation, or storage.

In this paper, we propose to solve such flexible image restoration problems with a novel multi-task learning framework, named prune-and-grow (PnG) network, where each task corresponds to fitting one of the requirements. As shown in Figure 1, unlike conventional multi-task networks [1, 16, 28, 34, 36] that consist of a common back-end trunk/modules with task-specific front-end branches, the proposed PnG network enables the sharing of partial parameters within the common back-end layers for different tasks, which computes common and task-specific features at various abstraction levels. Compared with conventional multi-task networks where each layer is either fully shared or fully task-specific, the in-layer partial feature sharing scheme of the PnG network gives a more general solution, which encourages more adequate and efficient information transfer and therefore improves the overall performance, as will be shown later in Sections 4 and 5.

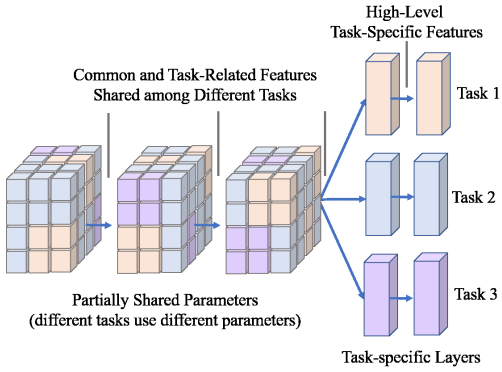
While related, the proposed PnG network is different from previous work like the slimmable networks [43, 44] or nested sparse networks [9, 22], where the latter always reuses the channels of the smaller models in the larger models and the former does not enforce such a constraint (in other words, PnG is a more general method). In fact, mathematically, the parameter sharing of the common trunk layers can be viewed as a generalized task assignment problem, where each parameter w can be assigned to an arbitrary number of k tasks. As shown in Figure 1a, in the conventional multi-task networks, $k=1$ or $k=n$ for all w in each layer. In the slimmable networks as illustrated in Figure 1b, parameters in different channels are sorted into c groups, and all parameters in the j-th group are assigned to k_j tasks



(a) Fully shared or task-specific layers



(b) Group-wise shared channels



(c) In-layer partial parameter sharing

Figure 1: Different multi-task network architectures. By in-layer partial parameter sharing, our method learns both common and task-related features at various abstraction levels. Different colors correspond to different tasks.

together, and $k_1 < k_2 < \ldots < k_c$. In contrast, the PnG network aims at solving the generalized assignment problem, as shown in Figure 1c. Since the generalized task assignment is NP-hard, the PnG framework seeks for an approximate solution iteratively.

Specifically, the PnG network learns the multi-task parameter sharing structure and parameter values in a progressive iterative manner. Starting from a model trained for one task, we prune the model weights while maintaining the reconstruction performance. Next we grow the pruned model back into a full model where we refill the pruned parameters by training for another task. As a result, the weight parameters of the first pruned model are for the inference of both tasks, while the re-grown parameters are for the second task only. By repeatedly performing the pruning and growing process, we can gradually tune different parts of the model parameters to accommodate more and more tasks.

Different from previous packable networks [29, 30] that are based on unstructured weight pruning, our prune-and-grow operates at the fine-grained micro-structure level, which benefits from the GEMM-based optimization for network inference, with little sacrifice of the task performance.

We apply the PnG framework for flexible image restoration, especially using NIC and SISR as two examples. We conduct extensive experiments over several benchmark datasets. Results show that compared with learning individual models for each task, the proposed method can reduce the model size by more than 50% and the computational cost by more than 25%, with almost no degradation over the image reconstruction quality.

2. Related Work

2.1. Model reduction & acceleration

The target of model reduction and acceleration is to simultaneously reduce the model size and accelerate inference computation, without sacrificing the performance of the original task. Various methods have been developed, including weight pruning [13, 14, 18, 23, 27, 40], weight quantization [41, 47], low-rank factorization [37], knowledge distillation [15], dynamic capacity networks [2]. Among all the efforts, weight pruning has been largely explored due to its model-structure-agnostic nature.

Unstructured pruning [13, 45] removes unimportant weight parameters that are randomly located in the weight tensor, which can hardly reduce inference computation in practice. The convolutional computation in DNN is commonly implemented as GEneral Matrix Multiplication (GEMM) by reshaping the weight tensors and input/output feature tensors to matrices [7]. In this context, the structured weight pruning is much more favored since it pursues both storage and computation efficiency by removing weight coefficients in a structured way that is beneficial for the GEMM computation. For example, unimportant channels or filters can be removed [14, 40] to directly reduce rows and columns in GEMM. However, removing large weight structures may have a severe impact on the original task performance. The work of [21] studied grouping and removing neighboring weights at a smaller block level, but only for fully connected layers. Recently a structured weight unification method has been developed as a less aggressive approach [18], where micro-structured 2D weight blocks or 3D weight cubes that align with GEMM are unified to have the same absolute value. A more balanced model can be obtained to preserve the task performance and reduce inference computation.

Inspired by the idea of using fine-grained micro-

structures in [18], we reduce the overall model size of multiple tasks by asking them to share model parameters at the level of micro-structures. Our PnG framework progressively trains the model instance to perfrom more and more tasks through iterative micro-structured weight pruning and weight growing. Model parameters are pruned and regrown based on fine-grained micro-structures, for a balanced between preserving individual task performance and reducing inference computation. Sharing model parameters at the level of micro-structures also encourages adequate sharing and prevents negative transfer.

2.2. Multi-task learning

The rationale of multi-task networks is the idea of sharing model parameters across multiple tasks. Based on the observation that bottom layers capture low-level features that can be shared across tasks, while top layers capture high-level features that are more task specific, most multi-task networks adopt a common architecture comprising of a set of feature extraction base layers that are shared across tasks followed by specialized network branches for task-specific inference, as illustrated in Figure 1 (a). For example, face detection, facial landmark localization and pose estimation are jointly explored in [34], and pose estimation and action recognition are jointly studied in [11]. Transfer learning like cross-residual learning [20] is also used to transfer knowledge among multiple tasks.

As the complexity of the tasks (so as the network architectures) increases, hand-designing the right network structure becomes very expensive. Some recent works investigate how to automate the process of such structure design. For example, the cross-stitching network [33] learns an optimal combination of shared and task-specific representations. A dynamic branch growing mechanism is developed in [28] to determine the network structure by taking into account both task relatedness and complexity of the model. Meta-learning is used by [16, 31] to dynamically predict the weights of the task-specific filters.

Ideally the common shared layers can extract common features that are generic enough and rich enough to support multiple tasks. However, it can be hard to learn a universal feature extractor in reality, especially for complicated tasks, due to the limited model capacity and limited training data. In such cases, negative transfer may happen where inadequate sharing may actually hurt the performance of individual tasks, such as the example shown in Figure 6. To alleviate the issue, one may need to increase the model complexity of the common feature extractor or add additional task-specific feature extraction layers, and therefore increase the required amount of training data and computation.

We extract both common and task-related features at various abstraction levels. Our PnG learning framework enables partial parameter sharing within the feature extraction layers. The in-layer sharing structure (*i.e.*, which parameters to be shared by which tasks) and the parameter values are automatically determined in training. The framework is task-generic and model-structure-agnostic and can be applied to various models for various applications.

The multi-task methods most related to our approach include slimmable networks [43, 44], nested structured sparse networks [9, 22], and nested unstructured sparse networks [29, 30]. Slimmable networks [43, 44] and nested structured sparse networks [9, 22] share channel groups among tasks, which is analogous to structured pruning of channel groups, and therefore usually cause large reconstruction performance drop for image restoration. Nested unstructured sparse networks like PackNet [30] and Piggyback [29] perform iterative unstructured pruning and retraining to pack multiple tasks. They can better maintain the task performance but cannot reduce inference computation. Moreover, these methods are seldomly applied to image restoration tasks, which usually require higher-quality features than classification or detection tasks.

2.3. Neural image compression

Neural image compression (NIC) has shown promising improvements for lossy image compression [3, 4, 26, 32]. NIC models feature a variational endoer-decoder (VAE) structure. An input image x is first encoded into a compact latent representation y by the encoder network, which is quantized into discrete $\hat{y}$ and further compressed losslessly into a bitstream with length $R(\hat{y})$. The decoder network then reconstructs a restored image $\hat{x}$ from $\hat{y}$. The reconstruction quality is measured by a distortion metric $D(x, \hat{x})$ (*e.g.* PSNR or MS-SSIM [39]). The rate-distortion (R-D) loss is used in training:

$$L = \lambda D(x, \hat{x}) + R(\hat{y}). \tag{1}$$

The hyperparameter λ controls the reconstruction quality, *i.e.*, a large λ results in smaller distortion but more bit consumption. Bitrate control remains one major issue of NIC, *i*.e., the capability to compress an image with varying qualities (different λs) based on practical needs. In general, NIC with each desired λ is treated as an individual task, where an NIC model instance needs to be trained and deployed. This can be too expensive for resource-limited applications.

There are some recent work exploring multi-rate NIC, such as the RNN-based models in a progressive encoding-decoding framework [19], which requires multiple iterations to obtain a high-quality image. The conditional autoencoder is also introduced that uses the Lagrange multiplier and quantization bin size as conditioning variables to control the compression quality [8]. A modulated generalized octave convolution is developed in [25] where the Lagrange multiplier is used to moldulate feature maps.

We apply our PnG framework to the multi-rate NIC application, where one multi-task model instance is learned to

support NIC with multiple compression qualities (*i.e.*, λs). Since our framework is model-structure-agnostic, we can flexibly accommodate any underlying NIC model structure and target loss, without requiring multiple iterations or specially designed conditional convolution or loss function.

2.4. Single image super-resolution

Single image super-resolution (SISR) aims at generating a high-resolution (HR) image y from its degraded low-resolution (LR) version x. DNN-based SISR methods have achieved great success in reconstructing visually accurate details. State-of-the-art SISR models, such as ESPCNN [35], EDSR [24], RDN [46], and WDSR [42], usually contain two parts: a feature extraction module that extracts rich feature maps from the LR input, followed by an upscaling module that aggregates the rich features to reconstruct the HR image. Since the shape of the upscaling module is related to the target scale factor (*e.g.*, 2×, 3×, or 4×), conventional SISR methods treat SR of each different scale factor as an independent task and learn an individual model instance for it. Recently, Meta-SR [16] uses meta-learning to dynamically predict the weights of the upscaling filters by taking the scale factor as input, which conducts SR of multiple scale factors with a single model instance.

We apply our PnG framework to SISR models to achieve multi-scale-factor SR effects. Similar to Meta-SR, the feature extraction module is shared by model instances of different scale factors. Different from Meta-SR where all scale factors use the same feature extractor as traditional layer-wise multi-task sharing, our method allows different scale factors to use different parts of the parameters of the feature extraction module, which generates task-specific features in early stages. Also, we assign a fixed set of weight coefficients for the upscaling module, instead of dynamically predicting the weights. Meta-SR is capable of coping with arbitrary non-integer scale factors. Our method can be generalized to include more scale factors progressively.

3. Methodology

We first formulate our problem. As illustrated in Figure 2, let $\{W_i\}$ denote a set of weight coefficients of a DNN model, where W_i represents the weight tensor of the i-th layer. Assume that we have n tasks $t_1, \ldots, t_n$, sharing the same set of weight tensors $\{W_i\}$. For each task t_j, a binary mask $M_{i,j}$ can be assigned to each weight tensor W_i, indicating which parameters in W_i are in effect for the inference of this task (with $M_{i,j}=1$) via the masked weight tensor:

$$\bar{W}_{i,j} = W_i \otimes M_{i,j}, \tag{2}$$

where $\otimes$ is element-wise multiplication. Theoretically, each parameter $w \in W_i$ can be assigned to an arbitrary number of k tasks, $1 \le k \le n$, but it is practically impossible to search

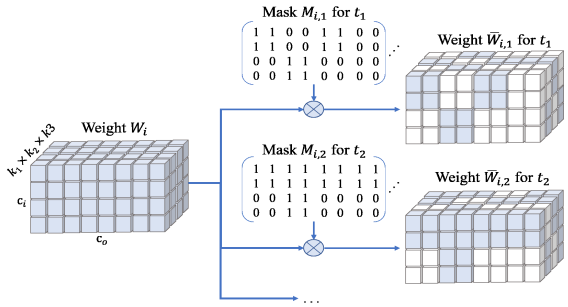


Figure 2: The reshaped weight tensor W_i is partitioned into micro-structured blocks, which is shared by multiple tasks. A binary mask $M_{i,j}$ is assigned to W_i for each task t_j indicating which parameters in W_i are used for the inference of this task.

for the optimal task assignment for all the parameters. Previous multi-task networks can be described as solving a highly reduced problem: parameters are assigned to tasks in the layer-wise fashion, and either $k=1$ or $k=n$. That is, parameters in W_i of a common shared layer are all assigned to all tasks ($k = n$), while parameters in a task-specific layer are all assigned only to one specific task ($k=1$). The slimmable networks [43, 44] or NestDNN [9] trains a single model that is executable at different widths, where parameters of different channels are sorted into c groups, and all parameters in the j-th group are assigned to the same k_j tasks. Channel groups of a narrower model is reused by a wider model, *i.e.*, $k_1 < k_2 < \ldots < k_c$.

In contrast, we aim at solving the generalized task assignment problem, which is in fact NP-hard. As shown in Figure 3, we propose a novel prune-and-grow (PnG) framework to find a suboptimal solution in a progressive iterative mannor. The basic idea is intuitive. Starting from a model instance trained for one task, we first prune the model weights to remove the unimportant weight parameters while maintaining the reconstruction performance of this task. Then we grow the pruned model back into a full model where we refill the pruned parameters by training for another task. As a result, the weight parameters of the first pruned model are used by both tasks. The regrown model parameters are for the second task only. By repeatedly conducting such pruning and regrowing process, we can gradually tune the model parameters part by part to complete more and more tasks.

3.1. Micro-structured weight pruning

As summarized in Section 2.1, we have multiple choices for weight pruning and growing. Unstructured pruning removes unimportant weight parameters that are randomly located in the weight tensor. Structured pruning removes entire structures like filters or channels. Unstructured pruning maintains the performance of the original model but can not speed up computation. Structured pruning can re-

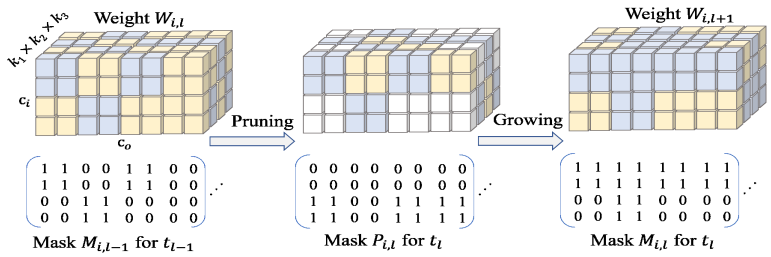


Figure 3: Illustration of the PnG framework. The current learned model instance $\{W_{i,l}\}$ and masks $\{M_{i,l-1}\}$ are used to learn masks $\{M_{i,l}\}$ through two iterative steps. a) pruning: micro-structured blocks are removed from $\{W_{i,l}\}$ while maintaining the original target performance of task t_l; b) growing: pruned blocks are refilled to regrow a full model towards task t_{l+1}.

duce inference computation by removing GEMM computing units, but usually hurts the original task performance. We exploit the micro-structure proposed in [18], which lies between the unstructured and structured pruning methods. We remove fine-grained structures, *e.g.*, $b \times b$ micro-blocks in the reshaped GEMM weight tensor, which can maintain the original task performance better than structured pruning and also reduce inference computation.

Specifically, for the i-th layer, weights W_i can be generally represented as a 5D tensor of size $(c_i, c_o, k_1, k_2, k_3)$, where c_i and c_o are the number of input and output channels, and (k_1, k_2, k_3) gives the kernel size, $k_3 = 1$ for 2D convolution and $k_2 = k_3 = 1$ for 1D convolution. The weight tensor can be reshaped as needed, resulting in equivalent matrix multiplciation with reshaped input and output tensors. Here, we reshape W_i into a 3D tensor of size $(c_i, c_o, k_1 \times k_2 \times k_3)$ as shown in Figure 2, *i.e.*, a number of $k_1 \times k_2 \times k_3$ matrices of size (c_i, c_o). We further partition each of the matrices into blocks of size $b_i \times b_o$. Weight coefficients are pruned in the block-wise mannor.

Blocking has been accepted as a general practice to speed up the GEMM computation in popular libraries like BLAS [10]. By removing block-wise micro-structures from the weight tensor, we can skip multiplications in GEMM computation. The larger the blocks, the more multiplications we can save per block, but the harder for the pruned model to maintain the original performance. Also, the smaller the block, the more flexible the method is to accommodate various model shapes. In this work, we simply select the smallest blocks size 2×2, and use a pruning ratio p to control the tradeoff between the original task performance and inference computation.

Note that when c_i or c_o can not be divided by 2, the corresponding block B along the boundaries will automatically reduce to size 2×1 or 1×2 or 1×1. In other words, 2×2 is the largest size for a block.

It is worth mentioning that selecting the right block size may further improve the overall performance. For example, DNN models for detection and classification often contains more redundancy than the restoration tasks. Larger block size can be used for detection and classification tasks. Also, models like MobileNet and SqueezeNet are more compact in design, and smaller block size may work better.

3.2. Multi-task prune-and-grow

Without loss of generality, assume that the n tasks $t_1, \ldots, t_n$ are ranked from easy (requiring simpler features) to hard (requiring richer features). The goal of the multi-task prune-and-grow is to learn a model instance with weights W_i^* for each layer as well as its associated binary mask $M_{i,j}$ for each task t_j, so that this model instance can conduct inference of each task t_j through masked weight tensor $\bar{W}_{i,j}$ of Eqn. (2). As illustrated in Figure 3, assume that we have the current model instance with weights $\{W_{i,l}\}$, and we have learned the binary masks $\{M_{i,1}\}, \ldots, \{M_{i,l-1}\}$ for the $l-1$ tasks before t_l. The current goal is to learn masks $\{M_{i,l}\}$ for task t_l, as well as updating the model instance. To achieve this, the PnG framework iterates the following two steps:

Pruning We first fix the weight parameters in $\{W_{i,l}\}$ that are masked by $\{M_{i,l-1}\}$ as being used by tasks before t_l. Since the binary masks are learned progressively, parameters fixed by masks for task t_{l-1} include those parameters that are fixed by masks for tasks earlier than t_{l-1}. These parameters will not be changed anymore in any of the remaining processing steps. Then we conduct the micro-structured pruning over $\{W_{i,l}\}$ to obtain pruned weight parameters $\{\hat{W}_{i,l}\}$ for task t_l.

Specifically, each weight tensor is partitioned into micro-blocks as described in the previous subsection. For the unfixed weight parameters in $W_{i,l}$, we can compute the pruning loss $L_p(B)$ for each of the unfixed micro-blocks B. Here we use the L_2 norm of the weight parameters in the block as $L_p(B)$. Then we rank these unfixed micro-blocks according to their pruning losses in ascending order, and select the top $p\%$ micro-blocks to be pruned. A binary pruning mask $P_{i,l}$ can be generated, where for an entry $p_{i,l,q} \in P_{i,l}$, $p_{i,l,q} = 1$ indicates that the corresponding weight $w_{i,l,q} \in W_{i,l}$ will be pruned (set to 0).

After that, we keep the weight parameters in $W_{i,l}$ that are masked by $P_{i,l}$ unchanged, and update the remaining

tunable weight parameters (neither masked by $P_{i,l}$ nor by $M_{i,l-1}$) through regular back-propagation by optimizing the loss function of task t_l. Finally, this pruning process will output the micro-structurally pruned weight parameters $\{\hat{W}_{i,l}\}$, which are the final model parameters used for the inference computation of task t_l.

Growing From the **Pruning** step, we have computed the micro-structurally pruned parameters $\{\hat{W}_{i,l}\}$ and the corresponding pruning masks $\{P_{i,l}\}$. Then the associated masks $\{M_{i,l}\}$ for the inference of task t_l can be obtained as:

$$M_{i,l} = M_{i,l-1} \cup \overline{P_{i,l}}, \tag{3}$$

where $\cup$ denotes the element-wise OR operation. Next, we fix the weight parameters in $\{\hat{W}_{i,l}\}$ that are masked by $M_{i,l}$ as being used by task t_l, and update the model to refill back the pruned weight parameters through regular back-propagation, towards optimizing the loss function of task t_{l+1}. This process will finally grow back a full model instance $\{W_{i,l+1}\}$. When the current task t_{l+1} is the last task t_n (*i.e.*, $l+1=n$), this full model instance $\{W_{i,l+1}\}$ is the final output $\{W_i^*\}$.

Algorithm 1 summarizes the entire training algorithm. It is easy to see that this PnG framework is task-generic and model-structure-agnostic, which can be applied to different types of networks for different tasks, without relying on hand-designed sharing and task-specific network structures. In the next sections, we apply this algorithm to two different representative use cases: NIC and SISR, which demonstrate the flexibility and effectiveness of our method. In nature, NIC aims at effective latent representation learning. The VAE structure is widly used as a feature embedding component in many other applications. SISR aims at realistic detail generation. It shares similar network structures with many other quality enhancement tasks.

4. Application: Multi-Rate NIC

As discussed in Section 2.3, it is generally difficult for NIC methods to control bitrates based on the R-D loss of Eqn. (1). Traditionally, multiple individual model instances need to be learned, one for each desired tradeoff factor λ. We apply our PnG network to learn the multi-rate NIC model, which largely reduces the required model size and also accelerate inference computation. The network architecture is shown in Figure 4.

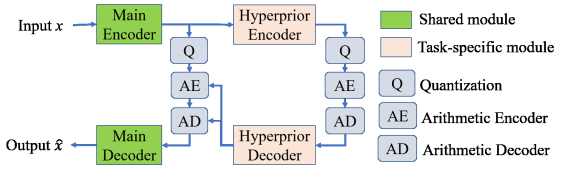


Figure 4: Architecture of our multi-rate NIC network.

Algorithm 1: Prune-and-grow training

Input : n tasks $t_1, \ldots, t_n$, the corresponding datasets $\mathcal{D}_1, \ldots, \mathcal{D}_n$ and loss functions $L_1, \ldots, L_n$, pruning ratio $p_1, \ldots, p_{n-1}$

Output: model instance $\{W_i^*\}$, masks $\{M_{i,1}\}, \ldots, \{M_{i,n}\}$

Learn $\{W_{i,0}\}$ by optimizing L_n over $\mathcal{D}_n$. Set $M_{i,0} = \mathbf{0}$.

for $l = 1$ **to** $n-1$ **do**

a) pruning

- Fix weight parameters in $\{W_{i,l}\}$ masked by $M_{i,l-1}$
- Prune the top p_l% unfixed blocks with minimum loss $L_p(B)$ and compute the pruning mask $P_{i,l}$
- Fix parameters masked by $P_{i,l}$ or $M_{i,l-1}$. Update the remaining parameters to get $\{\hat{W}_{i,l}\}$ by optimizing L_l over $\mathcal{D}_l$

b) growing

- Compute $M_{i,l}$ by Eqn. (3)
- Fix parameters in $\{\hat{W}_{i,l}\}$ masked by $M_{i,l}$. Regrow pruned parameters to get $\{W_{i,l+1}\}$ by optimizing L_{l+1} over $\mathcal{D}_{l+1}$

end for

return $\{W_i^*\} = \{W_{i,n}\}, \{M_{i,1}\}, \ldots, \{M_{i,n}\}$

4.1. Experimental setup

We use the scale hyperprior NIC method described in [4] as the base model, which is the most widely used NIC model in recent challenges and standardization activities, such as the IEEE MMSP Learning-based Image Coding Challenge 2020 [17]. Our experiments are based on the CompressAI PyTorch package [5]. The MMSP 2020 challenge provided the JPEG-AI benchmark dataset, comprising of 5264, 350, 40 training, validation, and test images, respectively, with resolutions ranging from 256×256 to 8K. This is one of the latest and largest benchmarks for NIC. CompressAI gives 8 pre-trained models corresponding to 8 compression quality levels, trained to optimize PSNR using loss of Eqn. (1) ($D(x,\hat{x})$ as MSE). We finetuned each pre-trained model using JPEG-AI training data with the corresponding λ setting. This gives the single-model PSNR-oriented results. In addition, we report MS-SSIM-oriented [39] results, where we started from the pre-trained PSNR-oriented model of the corresponding compression quality, and used the JPEG-AI training data to finetune the model parameters with the MS-SSIM-oriented loss function $L_{ssim} = \lambda(1-D(x,\hat{x}))+R(\hat{y})$, where $D(x,\hat{x})$ is MS-SSIM. This gives the single-model MS-SSIM-oriented approach.

The goal of multi-rate NIC is to reduce the overall model size and computation, without sacrificing the performance of individual tasks. In other words, the compression performance of single-model NIC is the target upper bound to achieve. We follow the convention of multi-task learning literature [1, 16, 31], and compare with single-model NIC to show that we can actually reach the target.

CompressAI uses a larger model for higher bitrate compression (quality 6 to 8) and a smaller model for lower bi-

Table 1: Number of parameters comparison for NIC.

config	individual×2	masked (2)	param reduction	individual×3	masked (3)	param reduction
big	23.63M	13.56M	43%	35.45M	15.30M	57%
small	10.15M	5.83M	47%	15.23M	6.59M	57%

trates (quality 1 to 5). The λs are ranked from small to large, corresponding to the tasks ranked from easy to hard. Based on these settings, we can form two sets of experiments:

- **2-model NIC**: 2 tasks (*i.e.*, λs) share the same model instance, which is called "masked (2)" in our results. We can have 3 sets of 2-task models for both PSNR and MS-SSIM-oriented results: PSNR: $\lambda = (0.0035, 0.0067), (0.013, 0.025), (0.0932, 0.18)$, SSIM: $\lambda = (4.58, 8.73), (16.64, 31.73), (115.37, 220)$.

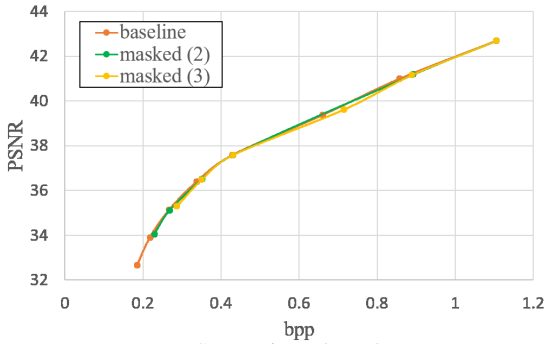
- **3-model NIC**: 3 tasks share the same model instance, which is called "masked (3)" in our results. We can have 2 sets of 3-task models for both PSNR and MS-SSIM-oriented results: PSNR: $\lambda = (0.0067, 0.0130, 0.0250), (0.0483, 0.0932, 0.18)$, SSIM: $\lambda = (8.73, 16.64, 31.73), (60.5, 115.37, 220)$

We empirically set pruning ratio for all λs as $p = 25\%$. Potentially for each task t_l one can search for the optimal prune ratio p_l by gradually increasing p_l (hence decreasing the task performance) until reaching a maximum tolerable performance loss. Here we take a one-fit-all simple setting for pruning ratio, which already performs well.

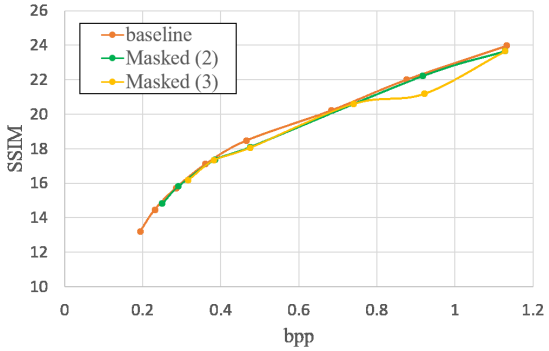
4.2. Results

Figure 5 (a) and (b) show the R-D curve of the PSNR-oriented and MS-SSIM-oriented results, respectively, over the JPEG-AI test set. From the results, by asking 2 λs to share the same model instance, the 2-model NIC has basically no performance drop comparing with the original single-model NIC, for both PSNR and MS-SSIM. For 3-model NIC, there is almost no performance drop over PSNR, and only very little drop in a short bpp range (from 0.7 to 1.0) over MS-SSIM. Note that here the MS-SSIM is magnified by taking its dB form since the original difference is too small to see clearly.

Table 1 shows the statistics of the number of parameters of the tested model instances. The high bitrate (big model) and low bitrate (small model) individual models have 11,816,323 and 5,075,843 parameters, respectively. The 2-model NIC can reduce the model size by 43% compared with using 2 individual models. The 3-model NIC can reduce the model size by 57% compared with using 3 individual models. The results demonstrate that the proposed PnG network can be effectively used for multi-rate NIC for model reduction and acceleration with almost no performance drop. For each single task, the maximum computation is the same as the original single model, which only happens for the last task added to the model. For all other tasks, the computation is reduced.



(a) PSNR-oriented results



(b) MS-SSIM-oriented results (magnified: $-10log_{10}(1-\text{SSIM})$)

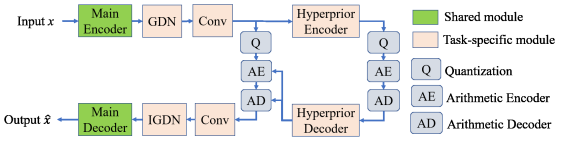
Figure 5: Performance of 2-model and 3-model NIC. The MS-SSIM is converted to dB to magnify the difference. There is no performance drop for 2-model NIC. 3-model NIC shows almost no performance drop over PSNR, and only a little drop in a short range of bpp for magnified MS-SSIM.

4.3. Ablation study

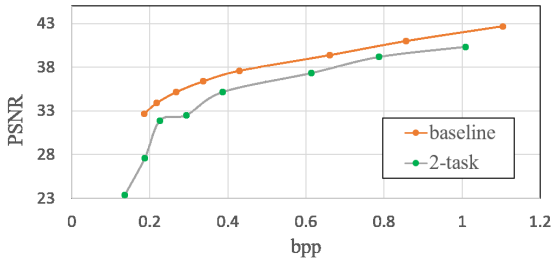
To further show the importance of sharing feature extractors at different abstraction levels, we also evaluated a traditional approach using the multi-task architecture of Figure 1 (a). Specifically, we form a 2-model NIC whose architecture is shown in Figure 6 (a) by asking 2 λs to share the same "Main Encoder" and "Main Decoder" as the common feature extractor, and adding 1 additional "GDN+conv" and 1 additional "conv+IGDN" component on the encoder and decoder side, respectively, as additional task-specific feature extractor. Figure 6 (b) gives the performance of this approach, which clearly shows that the common feature learner fails to learn a good general representation for both tasks. Due to inadequate sharing, it is hard for the additional task-specific layers to tailor the inefficient low-level common feature for conducting individual tasks.

Table 2: Performance comparison of multi-scale-factor SISR with individual models. GMAC is computed for 640×480 inputs.

scale	ssim		psnr		gmac		param	
	individual	3-scale	individual	3-scale	individual	3-scale	individual	3-scale
2×	0.9965	0.9964	36.38	36.06	1093	548	3,558,612	3,577,329
3×	0.9830	0.9819	32.63	32.24	1095	822	3,564,117	
4×	0.9708	0.9692	30.61	30.22	1097	1097	3,571,824	



(a) Architecture



(b) Performance

Figure 6: An example of an ineffective traditional multi-rate NIC method with layer-wise low-level common feature learning.

It is worth mentioning that we have also tried to apply the slimmable networks [44] for multi-rate NIC. However, removing channels, even only 10%, caused constant performance drop (as much as 2dB). This is consistent with previous literatures that the width of network (*i.e.*, the richness of feature) is critical for image restoration tasks [42].

5. Application: Multi-Scale-Factor SISR

As discussed in Section 2.4, most SISR methods train an individual model instance for each scale factor (2×, 3×, or 4×). We apply the PnG framework to achieve multi-scale-factor SISR, and the network architecture is shown in Figure 7. The feature learning module is shared by multiple scale factors with micro-structured masks, and each scale factor has its own task-specific upscaling module.

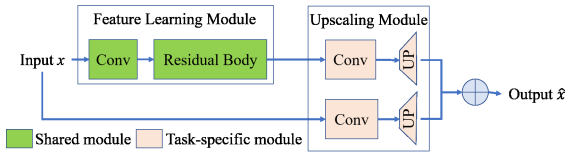


Figure 7: Architecture of our multi-scale-factor SISR network.

For experiments, we use the WDSR method [42] as our base model, which has shown state-of-the-art balanced performance in terms of both reconstruction accuracy and computation speed. We use the DIV2K benchmark dataset [38] released by the NTIRE 2017 Challenge, which contains 800 training images and 100 validation images. The validation set is used for test purpose here since the ground truth of the original test data is not available. Following the convention of previous literatures for SISR [38, 46], we only consider the PSNR and MS-SSIM on the Y channel of the transformed YCbCr color space. We empirically set $p = 50\%$ in this experiment.

Again, the goal is to reduce the overall model size and computation, and the performance of single-model SISR is the target upper bound to achieve. Following the convention of multi-task learning literature [1, 16, 31], we compare our method with single-model SISR.

Table 2 shows the PSNR and MS-SSIM comparison of our 3-scale-factor SISR model and the original single-scale individual models over the DIV2K validation set. From these results, by sharing the feature learning module, we reduce the model size by 67%, from requiring the total number of 10,694,553 parameters for 3 individual models to only 3,577,329 parameters. Also, we reduce the inference complexity by 50% and 25% for 2× and 3× cases. This is achieved by only slight sacrifice of the original reconstruction performance, *i.e.*, 0.3 dB for PSNR (1%) and 0.001 for MS-SSIM (0.1%).

6. Conclusion

We have proposed a novel PnG framework for learning multi-task networks, which seeks for a suboptimal solution of the generalized task assignment problem through iterative learning. The PnG network can partially share parameters within various feature learning layers among tasks to obtain both common and task-related features at different abstraction levels. The multi-task parameter sharing structure is automatically determined in an iterative progressive manner through micro-structured model pruning and growing. The fine-grained micro-structure is compatible with how matrix multiplication is performed in computer, so we can reduce the model size and also reduce the inference computation. The PnG framework is task-generic and model-structure-agnostic, and can be applied to different models of different tasks. Using image restoration, especially NIC and SISR, as two example applications, we have shown that our PnG network can largely reduce the overall model complexity of the state-of-the-art single-model-based methods, with almost no degradation of the original reconstruction performance.

References

[1] C. Ahn, E. Kim, and S. Oh. Deep elastic networks with model selection for multi-task learning. *ICCV*, 2019. 1, 6, 8

[2] K. Almahairi, N. Ballas, T. Cooijmans, Y. Zheng, H. Larochelle, and A. Courville. Dynamic capacity networks. *ICML*, pages 2549–2558, 2016. 2

[3] J. Ball, V. Laparra, and E. Simoncelli. End-to-end optimized image compression. *ICLR*, 2017. 1, 3

[4] J. Ball, D. Minnen, S. Singh, S. Hwang, and N. Johnston. Variational image compression with a scale hyperprior. *ICLR*, 2018. 1, 3, 6

[5] J. Bgaint, F. Racap, S. Feltman, and A. Pushparaja. Compressai: A pytorch library and evaluation platform for end-to-end compression research. *https://github.com/InterDigitalInc/CompressAI*, 2020. 6

[6] J. Chen, H. Chao, and M. Yang. Image blind denoising with generative adversarial network based noise modeling. *CVPR*, page 31553164, 2018. 1

[7] S. Chetlur, C. Woolley, P. Vandermersc, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. Cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014. 2

[8] Y. Choi, M. El-Khamy, and J. Lee. Variable rate deep image compression with a conditional autoencoder. *ICCV*, 2019. 3

[9] B. Fang, X. Zeng, and M. Zhang. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. *ACM Annual Int'l Conf. on Mobile Computing and Networking*, 2018. 1, 3, 4

[10] R. Geijn and J. Huang. How to optimize gemm. *https://github.com/flame/how-to-optimize-gemm/wiki*. 5

[11] G. Gkioxari, B. Hariharan, R. Girshick, and J. Malik. R-cnns for pose estimation and action detection. *arXiv preprint-arXiv:1406.5212*, 2014. 3

[12] S. Guo, Z. Yan, K. Zhang, W. Zu, and L. Zhang. Toward convolutional blind denoising of real photographs. *CVPR*, pages 1712–1722, 2019. 1

[13] S. Han, H. Mao, and W. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016. 2

[14] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. *ICCV*, 2017. 2

[15] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *NIPS Deep Learning and Representation Learning Workshop*, 2015. 2

[16] X. Hu, H. Mu, X. Zhang, Z. Wang, T. Tan, and J. Sun. Meta-sr: a magnification-arbitrary network for super-resolution. *CVPR*, 2019. 1, 3, 4, 6, 8

[17] IEEE and JPEG-AI. Jpeg-ai call for evidence. *IEEE MMSP2020 Challenge*, 2020. https://jpegai.github.io/. 6

[18] W. Jiang, W. Wang, and S. Liu. Structured weight unification and encoding for neural network compression and acceleration. *CVPR Workshop on Efficient Learning in Computer Vision*, 2020. 2, 3, 5

[19] N. Johnston, D. Vincent, D. Minnen, M. Covell, S. Singh, T. Chinen, S. Hwang, J. Shor, and G. Toderici. Improved lossy image compression with priming and spatially adaptive bit rates for re-current networks. *CVPR*, 2018. 3

[20] B. Jou and S.F. Chang. Deep cross residual learning for multitask visual recognition. *ACM Multimedia*, 2016. 3

[21] M. Keirsbilck, A. Keller, and X. Yang. Rethinking full connectivity in recurrent neural networks. *arXiv preprint arXiv:1905.12340*, 2019. 2

[22] E. Kim, C. Ahn, and S. Oh. Nestednet: Learning nested sparse structures in deep neural networks. *CVPR*, 2018. 1, 3

[23] Y. Li, S. Gu, C. Mayer, L. Van Gool, and R. Timofte. Group sparsity: The hinge between filter pruning and decomposition for network compression. *CVPR*, 2020. 2

[24] B. Lim, S. Son, H. Kim, S. Nah, and K. Lee. Enhanced deep residual networks for single image super-resolution. *CVPR*, 2017. 1, 4

[25] J. Lin, M. Akbari, H. Fu, Q. Zhang, S. Wang, J. Liang, D. Liu, F. Liang, G. Zhang, and C. Tu. Learned variable-rate multi-frequency image compression using modulated generalized octave convolution. *JPEG-AI MMSP*, 2020. arXiv preprint: 2009.13074. 3

[26] H. Liu, T. Chen, P. Guo, Q. Shen, X. Cao, Y. Wang, and Z. Ma. Non-local attention optimized deep image compression. *arXiv preprint arXiv:1904.09757*, 2019. 1, 3

[27] Z. Liu, H. Mu, X. Zhang, Z. Huo, X. Yang, T.K.T. Cheng, and J. Sun. Metapruning: Meta learning for automatic neural network channel pruning. *ICCV*, 2019. 2

[28] Y. Lu, A. Kumar, S. Zhai, Y. Cheng, T. Javidi, and R. Feris. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. *CVPR*, 2017. 1, 3

[29] A. Mallya, D. Davis, and S. Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. *ECCV*, 2018. 2, 3

[30] A. Mallya and S. Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. *CVPR*, 2018. 2, 3

[31] K. Maninis, I. Radosavovic, and I. Kokkinos. Attentive single-tasking of multiple tasks. *CVPR*, 2019. 3, 6, 8

[32] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool. Conditional probability models for deep image compression. *CVPR*, 2018. 1, 3

[33] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch networks for multi-task learning. *CVPR*, 2016. 3

[34] R. Ranjan, V. Patel, and R. Chellappa. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Transactions on PAMI*, 41(1):121–135, 2019. 1, 3

[35] W. Shi, J. Caballero, F. Huszar, J. Totz, A. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. *CVPR*, 2016. 1, 4

[36] X. Sun, R. Panda, R. Feris, and K. Saenko. Adashare: Learning what to share for efficient deep multi-task learning. *NeurIPS*, 2020. 1

[37] C. Tai, T. Xiao, Y. Zhang, X. Wang, and W. E. Convolutional neural networks with low-rank regularization. *ICLR*, 2016. 2

[38] R. Timofte, E. Agustsson, L. Van Gool, M. Yang, L. Zhang, B. Lim, S. Son, H. Kim, S. Nah, and K. Lee. Ntire 2017 challenge on single image super-resolution:methods and results. *CVPR Worshops*, 2017. 8

[39] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli. Image quality assessment: from error visibility to structural similarity. *TIP*, 13(4):600612, 2004. 3, 6

[40] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. *NIPS*, pages 2074–2082, 2016. 2

[41] S. Wu, G. Li, F. Chen, and L. Shi. Training and inference with integers in deep neural networks. *ICLR*, 2018. 2

[42] J. Yu, Y. Fan, J. Yang, N. Xu, Z. Wang, X. Wang, and T. Huang. Wide activation for efficient and accurate image super-resolution. *arXiv preprint:1808.08718*, 2018. 1, 4, 8

[43] J. Yu and T. Huang. Universally slimmable networks and improved training techniques. *ICCV*, 2019. 1, 3, 4

[44] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang. Slimmable neural networks. *ICLR*, 2019. 1, 3, 4, 8

[45] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. *ECCV*, pages 184–199, 2018. 2

[46] Y. Zhang, Y. Tian, Y. Kong, B. Zhong, and Y. Fu. Residual dense network for image super-resolution. *CVPR*, 2018. 1, 4, 8

[47] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *ICLR*, 2017. 2